社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2008－ARC－179 (2)
2008／8／5

ツインテール・アーキテクチャの評価

堀尾一生[†] 亘理靖展[††] 塩谷亮太[†]
五島正裕[†] 坂井修一[†]

本論文はツインテール・アーキテクチャの改良案を提案するものである．ツインテール・アーキテクチャには発行幅や命令ウィンドウ・サイズを実質的に増加させる効果があり，ウェイ数の大きなスーパスカラ・プロセッサの実現に貢献する技術である．本論文が新たに提案するハーフパンプ FU アレイは，ツインテール・アーキテクチャの消費電力を削減するための機構である．ハーフパンプ FU アレイはツインテール・アーキテクチャの命令のスループットを保ちながらも，消費電力を抑えることを可能にする．シミュレーションによる評価では，ハーフパンプ FU アレイを実装したツインテール・アーキテクチャは通常のツインテール・アーキテクチャと比べ 2.4%の性能低下にとどまり，ベースモデルのスーパスカラ・プロセッサに対して平均で 10.7%の性能向上が得られた．

Evaluation of Twin-tail Architecture

KAZUO HORIO,[†] YASUHIRO WATARI,[††] RYOTA SHIOYA,[†]
MASAHIRO GOSHIMA[†] and SHUICHI SAKAI [†]

This paper proposes a new inplementation of Twintail Architecture. Twintail Architecture is a technique which qives effect similar to increasing issue width and instruction window size, but at low hardware cost. It is expected to contribute to a practical implementation of ultra-wide super scalar processor. This paper's proposal, Half-pumped FU Array, reduces power consumption of Twintail Architecture at minimal performance cost. Our evaluation showed that Twintail Architecture with Half-pumped FU Array improves IPC of base model super scalar processor by 10.7%, a 2.4% performance loss from conventional Twintail Architecture.

1. はじめに

スーパスカラ・プロセッサの性能——IPC を向上させる一次的な方法は，そのウェイ数を増やすことである．しかし，IPC はウェイ数に比例して増加する訳ではない．その一方で，標準的なスーパスカラ・プロセッサのデザインでは，各部の回路規模はウェイ数のほぼ 3 乗に比例して増大してしまう[2]．そのため，ウェイ数を無闇に増加させると，回路面積は現実的な線を超えてしまう．したがって最近では，各コアのウェイ数は 2 程度に抑え，コアを多数並べることによって性能向上を目指すことが現実的な解だと認識されている[7]．しかし，そのようなアプローチで，いわゆる整数系のプログラムの性能向上を達成することは容易ではない．

これに対し本研究室では，回路規模がウェイ数の 3 乗には比例しないような技術をいくつも提案してきた．たとえば，スケジューリング・ロジックに対しては**マトリックス・スケジューラ**[3]，命令ウィンドウに対しては**ローカル・ディスタンス・ステアラ**[4]，リネーミング・ロジックに対しては**リネームド・トレース・キャッシュ**[5]，そして，レジスタ・ファイルに対しては**回路面積指向レジスタ・キャッシュ・システム**[6] を提案した．これらの技術の組み合わせによって，比較的ウェイ数の大きなスーパスカラ・プロセッサを現実的な面積で実現することが可能になると考えている．本稿で述べる**ツインテール・アーキテクチャ (Twintail Architecture)**[1] も，そのような技術の一つである．

本稿の提案手法

ツインテール・アーキテクチャには発行幅や命令ウィンドウ・サイズを実質的に増加させる効果があり，ウェイ数の大きなスーパスカラ・プロセッサの実現に貢献する．

ツインテール・アーキテクチャは，**IO tail/OoO tail** と呼ぶ 2 つの実行系を持つ．OoO tail は，通常のアウト・オブ・オーダ・スーパスカラ・プロセッサの命令ウィンドウ～バック・エンドと同じものと考えてよい．IO tail は，それに追加する形で配置される実行系である．フェッチされた命令は，IO tail/OoO tail のいずれかに送られ，実行される．

本稿では，この IO tail の動作周波数を半減する——**ハーフ・パンプ**化する技術を提案する．周波数を半減

† 東京大学大学院 情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo
†† ソニー 株式会社
Sony Corporation

する一方で，IO tail の演算器数を倍増することで IO tail のスループットは維持する．これによりわずかな性能低下で消費電力の削減を狙う．

以下，2 章でベースとなるツインテール・アーキテクチャについて説明した後，3 章でハーフ・パンプ化について述べる．ハーフ・パンプ化が IPC に与える影響については，4 章で述べる．

2. ツインテール・アーキテクチャ

2.1 ツインテール・アーキテクチャの構成

図 1 にツインテール・アーキテクチャのブロック図を示す．レジスタ・ファイルの直下に，命令ウィンドウと並列に演算器群を追加しているのがツインテール・アーキテクチャの特徴である．

追加された演算器群からなる実行系を **In-Order tail**，命令ウィンドウとその下流の演算器による実行系を **Out-of-Order tail** と呼ぶ．図 1 において，命令ウィンドウと並行に配置された 4 つの演算器が In-Order tail，命令ウィンドウと続く 2 つの演算器が Out-of-Order tail である．本論文では以降，In-Order/Out-of-Order tail をそれぞれ **IO tail/OoO tail** と略す．

IO tail の強みは，演算器とバイパス・ネットワークのみで構成されているという点である．このため従来のスーパスカラ・プロセッサの実行コアにはないスケーラビリティを持つ．

IO tail に演算器を追加しても，増加する回路面積はほぼ追加した演算器の分だけである．一方で，通常のアウト・オブ・オーダ・スーパスカラ・プロセッサの発行幅を増やそうとするとバックエンドの様々な機構に影響が出る．まず命令キューの読み出しポートが増加する．スケジューラのウェイクアップ信号が増加する．レジスタの書き込みポートが増加する．これらの機構の実体は多ポートの SRAM なので，その回路面積はポート数の 2 乗に比例する．このようにスーパスカラ・プロセッサの演算器を増やすということは大きなハードウェア・コストを伴う．

ツインテール・アーキテクチャは IO tail と OoO tail を併せ持つことによって，命令セットや周辺の回路のフレームワークを変えることなく，柔軟なスケーラビリティを実現する．

OoO tail は通常のスーパスカラ・プロセッサと同様の構成をとるため，特に解説すべき点はない．本節では以降，IO tail の動作と，IO tail/OoO tail への命令の振り分けについて説明する．

2.1.1 In-Order tail

IO tail は，アレイ状に配置された演算器群である．アレイの幅——列の数は，プロセッサのフェッチ幅と同じとする．同時にフェッチされた命令，すなわち同じ**フェッチ・グループ**内の命令は（互いの位置を替えることなく），フェッチ・グループ内の位置に対応した列に送られる．

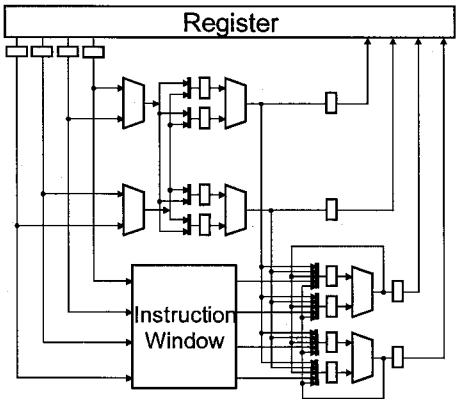

図 1 ツインテール・アーキテクチャのブロック図
Fig. 1 Block diagram of Twin-tail Architecture

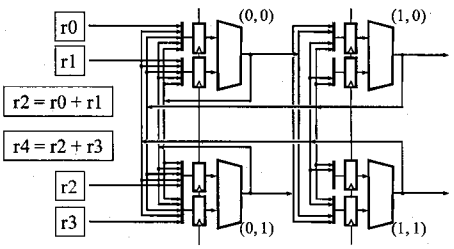

図 2 In-Order tail のブロック図
Fig. 2 Block diagram of In-order tail

IO tail は命令ウィンドウのような命令を溜めておく記憶領域を持たない．命令は毎サイクル次段の演算器に送られ，最終段を抜けると実行の有無に関わらず IO tail からいなくなる．

In-Order tail の動作

IO tail は到着した時点でソース・オペランドが揃っている命令を 1 段目の演算器で実行する．さらにアレイ状に配置された演算器間で実行結果の受け渡しを行い，IO tail で実行可能な命令を増やす．この実行結果の受け渡しについて，**図 2** を用いて説明する．

図 2 はフェッチ幅 2 × 2 段の IO tail に命令が到着した状況の例である．到着した 2 命令の間には依存関係があり，r4 = r2 + r3 は r2 = r0 + r1 の結果を用いる．また，ソース・オペランド r0，r1，r3 は到着した時点で既に値が得られているとする．ゆえに到着した時点で（1 段目の演算器で）命令 r2 = r0 + r1 は実行可能であるが，r4 = r2 + r3 は実行不可能である．この状況における IO tail の動作を時系列を追って説明する．

(1) 命令とそのオペランドが 1 段目の演算器の前のパイプライン・レジスタに入っている．

(2) 1 サイクル目，命令 r2 = r0 + r1 が演算器 (0,0) で実行される．実行結果 r2 はバイパス・ネット

ワークによって演算器群に伝搬する．同時に，実行されなかった命令 r4 = r2 + r3 が次段の演算器 (1,1) の前のパイプライン・レジスタに移動する．この時，既に得られていたオペランド r3 にはあえて ALU を通すため 0 を足す操作を行う．これにより実行結果を送るためのデータパスを流用して，使用されなかったオペランドを次段に送ることができる．r2 は命令 r4 = r2 + r3 の待ち望んだオペランドなので，選択されて演算器 (1,1) の前のパイプライン・レジスタに入る．

(3) 2 サイクル目，命令 r4 = r2 + r3 が演算器 (1,1) で実行される．

このように実行結果を次段の演算器に送ることで，同じフェッチ・グループ内の依存関係を解決し，IO tail で実行可能な命令を増やすことができる．

また後段の演算器からは，自分自身と前段の演算器まで実行結果を送るデータパスを設けた．これによりフェッチ・グループ間をまたがる依存関係を解決することができて，IO tail で実行可能な命令はさらに増加する．

In-Order tail で実行する命令

IO tail では，整数演算命令，分岐命令，ロード命令の実行を想定している．

Out-of-Order tail への実行結果のバイパス

IO tail で実行された命令の結果は OoO tail にバイパスする．IO tail に多段に配置された演算器の全てから OoO tail へのバイパスを行うと，OoO tail のバイパス・ネットワークが複雑になる．したがって，バイパス・ネットワークの複雑化を避けるため，OoO tail へのバイパスは，命令が IO tail を抜ける時に最終段の演算器から行う．また，OoO tail から IO tail へはオペランドのバイパスを行わない．

ロード命令の実行結果のバイパス

IO tail でアドレス計算を行ったロード命令は，次段の演算器から 1 次キャッシュへのアクセスを開始する．キャッシュは IO tail と OoO tail 両方からの要求を処理するため，調停回路を必要とする．IO tail で実行したロード命令の結果は IO tail ではなく，OoO tail へ直接送る．この結果は OoO tail からキャッシュ・アクセスを行った場合と同じパスを使ってバイパス，書き込みを行う．

2.1.2 命令の振り分け

IO tail/OoO tail への命令の振り分けは，”IO tail で実行できる命令は OoO tail へディスパッチしない”ポリシーで行う．

IO tail では，命令は実行可能/不可能に関わらず一定のサイクルで同じ演算器列を通り抜ける．つまり実行不可能な命令を IO tail へ投入することを止めても，実行可能な命令の使えるハードウェア資源は増加しない．したがって IO tail へ選択的な命令の投入を行っても性能が改善されることはない．

一方で，OoO tail においては命令は命令ウィンドウ・エントリなどの資源を占有する．したがって OoO tail へ選択的に命令を投入することは，OoO tail の資源を節約することにつながる．

In-Order tail における実行可能性の判定

IO tail における命令の実行可能性を調べるために，以下の動作をレジスタ・リネームと並列に行う．

まずレジスタ・リネーム時にソース・オペランドが揃っている命令は，そのデスティネーション・レジスタ番号をテーブルに書き込む．そして後続のフェッチ・グループの命令はそのテーブルを参照することで，自分のソース・オペランドが IO tail 内で供給されるか判断する．また同一フェッチ・グループ内の依存関係を調べるため，先行する命令のデスティネーション・レジスタ番号と後続の命令のソース・レジスタ番号を比較する．

実行可能性の判定の動作による遅延

レジスタの値とレディネスの情報を格納する領域は分離することができる．このような実装では，レディネスを調べるための遅延は値を読み出すための遅延よりずっと小さくなる．IO tail における実行可能性の判定には，ソース・オペランドのレディネスの情報が必要になるが，その値までは知らなくてもよい．従って IO tail における実行可能性の判定の動作がクリティカル・パスになることはない．

2.2 ツインテール・アーキテクチャの効果

ツインテール・アーキテクチャでは主に以下の 2 つの効果により，スーパスカラ・プロセッサのスループットを向上させる．

- 命令が早期実行される効果
- IO tail で実行される命令分，OoO tail の資源が節約される効果

本節ではこの 2 つの効果について説明する．

2.2.1 命令の早期実行による効果

IO tail はスケジューリング・ロジックを持たないため，OoO tail よりも命令の実行までのレイテンシが短くなる．**図 3** の上はツインテール・アーキテクチャのパイプライン，下はスーパスカラ・プロセッサのパイプラインである．IF はフェッチ，RF はレジスタ読み出し，disp はディスパッチ，sched はスケジューリング，issue は発行，exec は実行，WB は書き戻しの，それぞれのパイプライン・ステージを表している．

図 3 に示したのは命令 i1 の結果を命令 i2 が使用するようなプログラムの実行の様子である．今命令 i1 が IO tail で，命令 i2 が OoO tail で実行される状況を考える．命令 i1 が IO tail で実行され早期に結果が得られたことで，命令 i2 は命令ウィンドウに入り次第，発行される．一方，通常のスーパスカラ・プロセッサでは，命令 i1 の実行終了と共にジャスト・イン・タイムでその結果を利用できるように，命令 i2 は 1 サイク

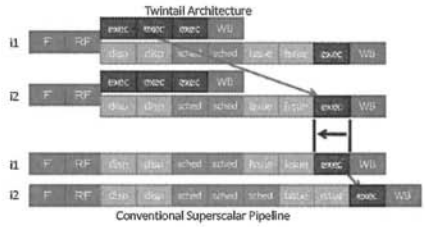

図 3 早期実行のパイプライン・チャート
Fig. 3 Pipeline-chart of pre-execution

ル遅れてウェイクアップされる．

このように通常のスーパスカラ・プロセッサならば命令を即座に発行できない状況でも，ツインテール・アーキテクチャでは命令を発行できる可能性がある．ここに示した例では，ツインテール・アーキテクチャは命令列全体の実行を 1 サイクル短縮している．

また，特にロード命令と分岐命令は早期実行の効果が大きい．

- ロード命令が早期実行される効果
  キャッシュ・アクセスのレイテンシは長いため，IO tail の実行タイミングで開始されると，レイテンシを隠蔽できる効果が高い．
- 分岐命令が早期実行される効果
  IO tail で分岐命令の結果が得られると，分岐予測ミスの発覚が早くなるため，分岐予測ミス・ペナルティが軽減される．

2.2.2 Out-of-Order tail の資源が節約される効果

2.1.2 で述べたように，IO tail で実行可能な命令は OoO tail にディスパッチしない．そのため IO tail で実行される命令の分だけ OoO tail の資源が節約できる．具体的には IO tail の存在により，発行幅と命令ウィンドウ・サイズが実質的に増加したような効果が得られる．

この内，特に命令ウィンドウ・サイズが実質的に増加する効果は性能に大きな影響を与えることが分かっている．

命令ウィンドウ・サイズが性能に与える効果

命令ウィンドウ・サイズは，キャッシュ・ミスが頻繁に発生するプログラムのパフォーマンスに大きな影響を与える．

ロード命令がキャッシュ・ミスを起こすと，そのロード命令がリタイアするまで後続の命令がリタイアできないので，命令ウィンドウが満たされた後，フロントエンドからの命令の供給が止まる．

図 4 にキャッシュ・ミスを起こすロード命令が近くに配置されたプログラムの実行の様子を示す．図 4 の上は，命令ウィンドウ・サイズに余裕がある場合で，下は余裕がない場合である．

命令ウィンドウ・サイズに余裕がある場合は，1 つ目のロード命令のメモリ・アクセスと並列に，2 つ目のロード命令のメモリ・アクセスが行える．命令ウィンドウ・サイズに余裕がない場合は，1 つ目のロード命令がリタイアするまで後続のロード命令が命令ウィンドウに入れず，メモリ・アクセスを開始できない．つまり，メモリ・アクセスが並列に行えなくなる．

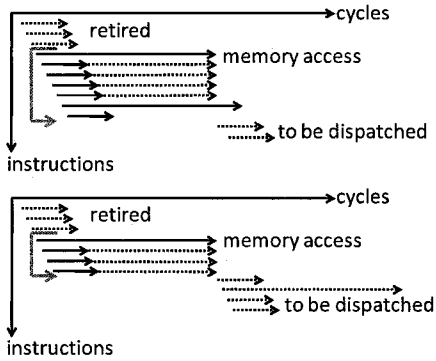

図 4 命令ウィンドウ・サイズがメモリ・アクセスの並列性を制限する
Fig. 4 Size of instruction window limits parallel memory accesses

3. 提案手法

本研究では，アレイ状に配置された演算器群を **FU アレイ**(Function Unit Array) と呼んでいる．本章ではツインテール・アーキテクチャの実装の選択肢として新たに提案する，**ハーフパンプ FU アレイ**について説明する．

ツインテール・アーキテクチャでは多くの命令を IO tail で実行することができ，実行した命令は OoO tail へディスパッチしないため，OoO tail の資源を節約できる．したがって IO tail のコスト・パフォーマンスを上げれば，性能を保ったまま OoO tail を縮小することができると考えられる．

そこで，我々は IO tail の消費電力を下げるため，ハーフパンプ FU アレイを提案する．ハーフパンプ FU アレイでは IO tail を OoO tail の周波数の半分，ウェイ数を倍にして動作させる．電力消費は周波数の 3 乗に比例するので，IO tail のスループットを保ったまま消費電力を削減することができる．

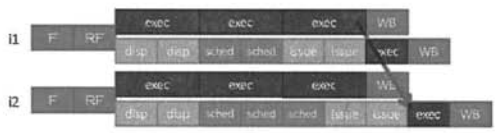

図 5 ハーフパンプ FU アレイのパイプライン・チャート
Fig. 5 Pipeline-chart with Half-pumped FU Array

図 5 に，3 段の IO tail においてハーフパンプ FU アレイを用いた場合の動作を示す．

各段に 2 サイクルかかるので，IO tail を抜けるの

に6サイクルかかる．命令は連続した2サイクル分のフェッチ・グループを合わせて，2サイクルに一度IO tailに投入する．OoO tailへのディスパッチは通常通り毎サイクル行う．

ハーフパンプFUアレイのデメリット

ハーフパンプFUアレイを用いることのデメリットとして以下の点があげられる．

- Out-of-Order tailへ結果のバイパスが遅れる
  OoO tailへのバイパスはIO tailを抜ける時に行われる．したがって，ハーフパンプFUアレイにすると，命令がIO tailを抜けるまでにかかるサイクル数が倍になるため，OoO tailへのバイパスが遅れることがある．
- In-order tail内での実行が遅れる
  ハーフパンプFUアレイを用いるとIO tail内での実行レイテンシが伸びる．したがって，分岐命令やロード命令の実行が遅れるため，2.2.1で述べたような，分岐予測ミス・ペナルティが軽減される効果や，ロード命令のキャッシュ・アクセス・レイテンシの隠蔽による性能向上の効果が減る．
- IO tail内で実行される命令数が減る
  同一フェッチ・グループ内に依存関係があると，コンシューマ命令の実行はプロデューサ命令の実行より1段遅れる．ハーフパンプFUアレイではフェッチ・グループが統合され，同一フェッチ・グループ内の依存が増えるため，IO tailを抜けるまでに実行不可能な命令が増加する．

次章ではハーフパンプFUアレイがツインテール・アーキテクチャの性能に与える影響について評価を行う．

4. 評　　価

4.1 評価方法

当研究室で開発したシミュレータ「鬼斬2」に対して提案手法を実装し，評価を行った．ベンチマーク・プログラムとして，CINT2000，CFP2000の先頭の1G命令をスキップした後の100M命令を用いた．表1にベースモデルの主要なパラメータを示す．

評価対象のツインテール・アーキテクチャは以下のような構成とした．

- **BASEMODEL**
  表1に示した構成のスーパスカラ・プロセッサ
- **TWINTAIL**
  フェッチ幅4×3段のIO tailを用いたツインテール・アーキテクチャ
- **HALFPUMP**
  IO tailに8×3段のハーフパンプFUアレイを用いたツインテール・アーキテクチャ

なお評価対象のツインテール・アーキテクチャでは，IO tail内のバイパスは3段目から1段目へのバイパスに1サイクルかかり，他は後続の命令が即座に利用可能とした．

4.2 IPC

図6に結果を示す．グラフはベース・モデルのIPCを1として正規化してある．

4.2.1 ツインテール・アーキテクチャの性能向上

BASEMODELに対し，TWINTAILでは最大で31.7%，平均で13.6%の性能向上が得られた．

4.2.2 ハーフパンプFUアレイの性能低下

HALFPUMPでは，BASEMODELに対し最大で29.7%，平均で10.7%の性能向上が得られた．TWINTAILに対する性能低下は平均で2.4%に留まった．

TWINTAILに対する性能低下は187.facerecが14.8%と最も大きかった．このベンチマークはTWINTAILで最もベース・モデルに対する性能向上が大きい．そのためハーフパンプFUアレイを導入することによってIO tailで早期実行できなくなる命令も多くなり，性能低下が大きいと考えられる．

4.2.3 IO tailで実行された命令の割合

ハーフパンプFUアレイの性能低下が平均して2.4%に留まった理由を考察するため，全体の内IO tailで実行された命令の割合を測定した．**図7**にIO tailで実行された命令の割合を示す．

ハーフパンプFUアレイではIO tailで実行できる命令が平均で5%程度減少しているが，それでも平均40%以上の命令がIO tailで実行されている．IO tailで実行できる命令は減少したが，命令全体の中の大きな割合がIO tailで実行できていることに変わりはなく，従ってハーフパンプFUアレイによる性能低下は低く抑えられていると言える．

5. ま と め

本論文は，発行幅を増やさずにスーパスカラ・プロ

表1 ベース・モデルの主要なパラメータ
Table 1 Principal parameters of the base model

Fetch Width	4
Issue Width	4
Integer Units	ALU × 4, MUL × 1, DIV × 1, 32entries instruction queue(used universally by all integer instructions)
Floating Point Units	ADD/MUL/DIV × 1, 16entries instruction queue(used universally by all FP instructions)
Memory Instructions Queue	16entries
L1 Cache	32KB, 4ways, 3cycles to access
L2 Cache	4MB, 8ways, 10cycles to access
Memory	200cycles to access
Physical Registers	int: 96 float:64
Re-order Buffer Entries	128
Pipeline Depth	Fetch-3, Read-2, Dispatch-2, Schedule-2, Issue-2, Write Back-2

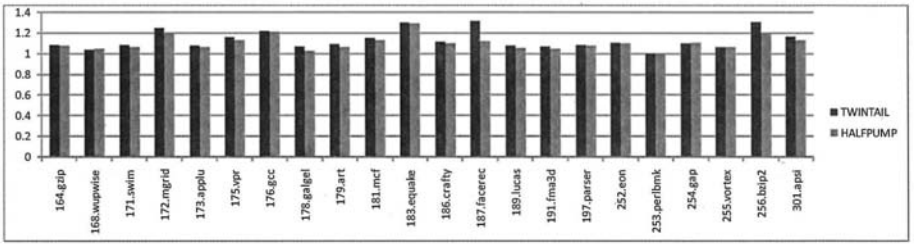

図 6 ツインテール・アーキテクチャの IPC
Fig. 6 IPC of Twin-tail Architecture

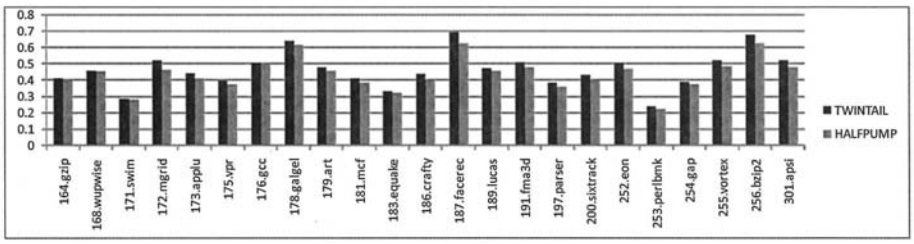

図 7 IO tail で実行された命令の割合
Fig. 7 percentages of instructions executed at In-Order tail

セッサに演算器を追加し，実行可能な命令数を増加させるツインテール・アーキテクチャについて説明した．また追加した演算器を半分の周波数で動作させることで消費電力の低減を図る，ハーフパンプ FU アレイを提案・評価した．

シミュレーションによる評価では，ツインテール・アーキテクチャはベース・モデルのスーパスカラ・プロセッサに対し平均で 13.6%の性能向上が得られることを示すことができた．またハーフパンプ FU アレイを用いた場合の性能向上は平均 10.7%であり，標準的なツインテール・アーキテクチャの構成に対して 2.4%の性能低下に留まることを示すことができた．

今後の研究方針として，プロセッサ内での IO tail の役割を高めていくことを考えている．平均 40%もの命令が IO tail で実行できるため，OoO tail の命令ウィンドウ・サイズや発行幅を縮小した場合の性能に対する影響も小さいと考えられるからである．

また IO tail にハーフパンプ FU アレイを用いることで，IO tail での命令実行に余裕が生まれるため，0 次キャッシュを設けることができるようになる．これにより，ロード命令に依存する命令も IO tail で実行することで，さらに多くの命令を IO tail で実行し，OoO tail の資源を更に節約することができるようになると考えている．

今後もより現実的な考察を重ね，魅力的な選択肢としてツインテール・アーキテクチャを提案していきたい．

参 考 文 献

1) 堀尾一生，平井遥，五島正裕，坂井修一: ツインテール・アーキテクチャ，先進的計算基盤システムシンポジウム *SACSIS2007*, pp.303-311, May, 2007
2) 五島正裕: *Out-of-order ILP* プロセッサにおける命令スケジューリングの高速化の研究，京都大学 *(博士論文)*, March, 2004
3) M. Goshima, K. Nishino, Y. Nakashima, S. Mori, T. Kitamura, and S. Tomita: *A High-Speed Instruction Scheduling Scheme for Superscalar Processors*, *MICRO-34*, pp. 225-236 (2001)
4) 服部 直也，高田 正法，岡部 淳，入江 英嗣，坂井 修一，田中 英彦: 発行時間差に基づいた命令ステアリング方式，情報処理学会論文誌 コンピューティングシステム *(ACS-7)*, pp.80-93, Oct 2004.
5) 一林 宏憲，塩谷 亮太，入江 英嗣，五島 正裕，坂井修一: 逆 *Dualflow* アーキテクチャ，先進的計算基盤システムシンポジウム *SACSIS2008*, pp.245-254
6) 塩谷 亮太，入江 英嗣，五島 正裕，坂井 修一: 回路面積指向レジスタ・キャッシュ，先進的計算基盤システムシンポジウム *SACSIS2008*, pp.229-236
7) Karthikeyan Sankaralingam, Ramadass Nagarajan, Haiming Liu, Changkyu Kim, Jaehyuk Huh, Doug Burger, Stephen W. Keckler, Charles R. Moore: *Exploiting ILP, TLP and DLP with the Polymorphous TRIPS Architecture ISCA 2003* pp.422-433